A Brief Introduction to *MATLAB*®

References: The Student Edition of MATLAB® Version 4 and 5 User's Guides
Introduction to MATLAB® 6 by DM Etter and DC Kuncicky, with D Hull
Feedback Control of Dynamic Systems, 4th Ed., by GF Franklin, JD Powell, and A Emami-Naeini

---

The *MATLAB* User Interface

You launch *MATLAB* from either the computer desktop icon or the `>Start>Programs>Matlab 6.x>Matlab 6.x` pull-up list. The launch may be delayed due to virus-scanning, so be patient.

The MATLAB window will be displayed, and contain three major sub-windows labeled: `Command Window` along the right side, `Workspace/Launch Pad` in the upper-left corner, and `Current Directory/Command History` in the lower-left corner.

Change the current directory from the default to `E:\Student Files`, so that you may easily access your data/script files. This change can be done from either the MATLAB window toolbar or the `Current Directory` window toolbar.

Much of the following "tutorial" will focus our attention on typing script commands within the `Command Window` at the user-command prompt indicated by the >> symbol. Additional `Figure` windows may pop-up when plotting of data occurs.

Vector definition and addressing

Explicit definition

```
»x=[0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0] % commas
```

or

```
»x=[0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0] % spaces
```

Implicit definition (colon notation)

```
»x=(0:0.1:1.0)  % format: x=(start_value:D_value:end_value)
```
or
```
»x=linspace(0,1.0,11)
           % x=linspace(first_value,last_value,# of values)
```
or
```
»x=logspace(0,2,100); % the ; suppresses on-screen echo
  % x= logspace(first_exponent,last_exponent,# of values)
  % here the the values go from 10^0 to 10^2 with 200
  % increments inclusive
```

Vector element addressing

```
»x(4) % 4th element of x
```
or
```
»x(1:5) % elements 1 through 5  of x
```
or
```
»x(1:3:10) % increment by 3 from element 1 through 10  of x

»x=(0:0.1:1.0)    % generates a row vector

                  % transposing x will give a column
                  % vector

»y=x' % the ' operator transposes an array (using .'
      % operator gives the non-complex-conjugate transpose)
```
or
```
»y=[1;2;3;4;5;6;7;8;9;10]   % here the ; separating the
                            % elements inserts the next
                            % element on a new row.
                            % Hence, y is a column vector.
```

Array definition and addressing

```
»A=[1,2,3,4,5;6,7,8,9,10]
```

The semi-colon separates the first row from the second row. Each row has 5 elements separated by commas.

or
```
»A=[1 2 3 4 5 <CR> 6 7 8 9 10 ]
```

The <CR> separates the first row from the second row. Each row has 5 elements separated by spaces.

```
»A(2,4) % 4th element of row 2 of A
»A(1:2,2:3)  % sub-array of A containing elements in the
             % first two rows of A in columns 2 and 3
```

Note: given an array **A**

```
»A.^3   % raises each element of A to the power of 3
        %(Note the period before the ^)

»A^3    % raises matrix A to the power of 3

»whos   % displays the variables and their size
```

2D Plotting

```
»x=(0:pi/100:2*pi); % create 201 x values from 0 to 2π
»y=sin(x);
»plot(x,y) % single plot on a single set of axis
```

or

```
»figure(2)        % creates second figure window
»z=cos(x*4);
»plot(x,y,x,z)    % two plots on a single set of axis
                  % (Note that they both share the same
                  % independent variable, x)
```

Axis labeling

```
»xlabel('x axis')
»ylabel('y axis')
»title ('My title')
»grid
```

or, use a “command cascade”

```
»xlabel('new x'),ylabel('new y'), title ('new title'), grid
```

Polar plots can be obtained as follows

```
»figure(3)        % creates third figure window
»theta=x;    % theta is the polar "angle" in radians
»sigma=y;    % sigma is the polar "radius"
»polar(theta,sigma)    % plot sin(x) vs x in polar coords
```

2D Sub-Plotting

To create multiple plot windows within a single figure, use the `>>subplot` command. As an example, generate a quadratic polynomial to be displayed four different ways

```
»clc         % clears the "command window"
»clf         % clears the current figure

»clear  % THIS COMMAND NUKES ALL VARIABLES FROM MEMORY
        % there is no "Are you sure? <y or n>" safety net


% generate new data

»x=(0:0.5:50); % create 101 x values from 0 to 50
»y=5*x.^2;
```

(continued)

```
»subplot(2,2,1), plot(x,y),
        title('Polynomial – linear/linear axes'),
        ylabel('y'),grid
»subplot(2,2,2),semilogx(x,y),
        title('Polynomial – log/linear axes'),
        ylabel('y'), grid
»subplot(2,2,3),semilogy(x,y),
        title('Polynomial – linear/log axes'),
        xlabel('x'),ylabel('y'), grid
»subplot(2,2,4),loglog(x,y),
        title('Polynomial – log/log axes'),
        xlabel('x'),ylabel('y'), grid
```

Note that `»subplot (u,v,w)` sets up a `u-by-v` "plot matrix" within the figure window. Each respective sub-figure is addressed by parameter `w` starting at the upper-left position in the grid.

Data Formats

Display formatting is by default "short" which means fixed-format with 4 decimal digits, and with "loose" organization that employs extra line-feeds when displaying matrix contents. These conditions are specified within the MATLAB "Command Window" via the `>File>Preferences` pull down menu. To change these preferences from the command line specify

```
»format short     % example 16.1234 (4 decimal digits)

»format long      % example 16.12345678901234
                  % (14 decimal digits)

»format short e   % e-format 1.6123e+01

»format long e    % e-format 1.612345678901234e+01

»format blank     % 16.12 (2 decimal digits)

»format compact   % reduces extra line-feeds in matrix
                  % displays

»format loose     % returns extra line-feeds in matrix
                  % displays
```

Differential equations

Suppose we have the following differential equation

$$\ddot{x} + \dot{x} + x = f(t) \tag{1}$$

where

$f(t)$ = unit step input

Note

$$\ddot{x} + 2\zeta\omega_n \dot{x} + \omega_n^2 x = f(t)$$

therefore in the above example

$\omega_n = 1$ and $\zeta = 0.5$

hence the system is underdamped!

1. convert Eq.(1) in *state-space equation* format

Let $x_1 = x$

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= f(t) - x_1 - x_2 \end{aligned} \tag{2}$$

2. putting Eq.(2) into *state-space vector* (ss) format

$$\{\dot{x}\} = [A]\{x\} + [B]\{u\}$$

$$\{y\} = [C]\{x\} + [D]\{u\}$$

we get

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} f(t)$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} f(t)$$

3. Eq.(1) in *transfer function* (tf) format

$$\frac{X(s)}{F(s)} = \frac{1}{s^2 + s + 1}$$

for f(t) = unit step input

$$X(s) = \frac{1}{s^2 + s + 1}\left(\frac{1}{s}\right)$$

Solution scenarios

1. Using the **Runge-Kutta method**

Define a *function m-file* from the MATLAB Window toolbar by choosing **File** menu, then **New**> m-**File**, then type the following

```
function xdot = xdeqns(t,x);
xdot = [x(2)
        1-x(1)-x(2)];        % xdot equation column
                             % vector
```

Once your function is entered choose **Save** from the **File** menu and *accept* the default name.

To solve the function use the Runge-Kutta solver built within MATLAB called **ODE23** or **ODE45**

- generic format `»[t,x]=ode45('function name',tspan,y0);`
- for our example

```
»tspan=[0 10];
```

Start time Stop time

```
»y0=[0;0];
```

$x(1)_{IC}$; $x(2)_{IC}$

```
»[t,x]=ode45('xdeqns',tspan,y0);
```

The ODE solver returns a time vector, t, and a solution array, x. To parse the solution array

```
»x1=x(:,1);  % 1st column x(1)
»x2=x(:,2);  % 2nd column x(2)
»figure(1)   % define active figure window
»plot(t,x1,t,x2,'--') % where '--' represents line style
```

Alternatively, you may define the A, B, C, D arrays from the **state space** representation

```
»A=[0 1;-1 -1];

»B=[0;1];

»C=[1 0;0 1];

»D=[0;0];

»sys=ss(A,B,C,D);   % state vector system definition
```

```
»t2=linspace(0,10,100);       % generate a time vector

»[yy,xx]=step(sys,t2);        % simulate the system using a
                              % unit step input
»yy1=yy(:,1);

»yy2=yy(:,2);

»figure(2)          % define second figure window

»plot(t2,yy1,t2,yy2)
```

2. Solution via Laplace transforms in the **s-domain**

```
»num=[1];           % 0s^2+0s+1

»den=[1 1 1];       % 1s^2+1s+1

»sys=tf(num,den);   % transfer function system definition

»t3=linspace(0,10,100);

»[yyy,xxx]=step(sys,t3);

»figure(3)          % define third figure window

»plot(t3,yyy,'+')
```

3. Solution via "block diagrams" using the *MATLAB* add-on program called *SIMULINK*®.

In brief, the *SIMULINK* environment is a graphical approach to system definition, response simulation, and graphing. And simulation data may be routed from the *SIMULINK* environment to the *MATLAB* `Command Window` for further analysis, graphing, export, etc.

To use *SIMULINK*® see the separate tutorial.

Further Scripting

The Runge-Kutta method of differential equation solution noted previously employed a *MATLAB* "function" or "*m*-file" to hold important state-equations (i.e., *xdeqns.m*). This function was repeatedly called from within "master" solution routine (i.e., `ode23`*.m*) sever dozen to several thousand times. We might also create *m*-files to assist other calculations and save ourselves repeated keyboard entries within the `Command Window`.

By way of example, suppose we wish to create our own solution to evaluate the "sinc" function $\sin(x)/x$ which we will call "*sinc_x.m*". (Note here that the *MATLAB* "canned" routine `sinc`*.m* is available to evaluate the function $\sin(\pi x)/\pi x$, and we could have employed `sinc.m` pre-scaling our input vector *x* by $1/\pi$.) In crafting our *m*-file *sinc_x* note that the function $\sin(x)/x$ may introduce a divide-by-zero error if the vector *x* contains one or more zero values. Hence, we will need to employ the *MATLAB* logical functions `find.m` and `abs.m` along with Boolean arguments for "less-than" (<) and "greater-than-or-equal-to" (>=) within our script-file.

From the *MATLAB* Window toolbar choose the **File** menu, then **New**> m-**File**

```
function s = sinc_x(x);
%
%SINC_X computes the values of (sin(x))/x
%
s=x;
set1=find(abs(x) < 0.001);  % abs(x) computes magnitudes
                            % of input vector elements,
                            % find()returns a vector of
                            % indices of the nonzero
                            % elements of the argument
                            % vector (),
                  % here that is the elements of x near
                  % zero (0.001) magnitude
set2=find(abs(x) >= 0.001); % elements of x with magnitudes
                            % greater than zero (0.001)
s(set1)=ones(size(set1));   % ones() creates a vector
                            % containing all unit values
                            % at matched indices where the
                            % vector x is near zero mag
       % Note via L'Hopital's Rule (sin(x))/x =1 for x=0
       %
s(set2)=sin(x(set2))./x(set2);
                            % evaluates (sin(x))/x at
                            % indices where x is non-zero
       % Note that the entire vector s is returned by
       % our sinc_x.m routine to the calling routine or
       % command line
```

To utilize `sinc_x.m` from the *MATLAB* `command line` enter the following

```
»clf
»x=(-15:0.1:15);
»y=sinc_x(x);
»plot(x,y), xlabel('x'),ylabel('y')
»title ('My Sinc Function'), grid, pause
  % pause requires user <cr> to return control
```

Saving Data to File

To create an ***Excel*** or ***Wordpad*** readable 2-column data file from ***MATLAB*** commands:

Using *MATLAB* <u>row-dominant</u> data vectors

```
% create x and y data

»xr=[0:0.1:10];        % independent variable
»yr=sin(2*pi*xr);      % dependent variable
»figure(1)             % first figure window
»plot(xr,yr)

% to combine xr and yr data as a matrix with
% synchronous x y values
```

% in adjacent columns, such as $\begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix}$, we need to define a

```
% new array

»z=[xr;yr]';       % where the apostrophe
                   % signifies "matrix transpose"

% save the z array to the file "myfile1.xls" in
% the "current directory" as a two-column,
% tab-delimited ASCII readable file suitable for
% direct Excel open/importation

»save('myfile1.xls','z','-ascii','-tabs');
```

Using *MATLAB* <u>column-dominant</u> data vectors

```
% create  x and y data
```

```
»xc=[0:0.1:10]';       % note the apostrophe
»yc=sin(2*pi*xc);
»figure(2)             % second figure window
»plot(xc,yc)

% to combine the xc and yc data as a matrix with
% synchronous x y values in adjacent
% columns (as above), we need another new array

»zz=[xc,yc];      % note change in syntax from above
                  % formulation for array  "z"

% save the zz array to file "myfile2.xls" in
% the "current directory" as a  two-column,
% tab-delimited ASCII readable file suitable for
% direct Excel open/importation

»save('myfile2.xls','zz','-ascii','-tabs');
```

Loading Data from File

It is possible create a ***MATLAB*** "workspace" variable from data contained in an ascii, tab delimited file created in ***Excel***, ***Wordpad,*** or ***LabVIEW.*** The file must contain *numbers only* without alpha headers and reside in the "current directory".

To import data from a file using a *'.txt'* extension use the following ***MATLAB*** command example:

```
% save the data from the file "myfile2.txt" to the array v:

»v=load('myfile1.txt');
```

To import a ***LabVIEW*** data file (no header) using a *'.lvm'* extension use:

```
% save the data from the file "myfile42.lvm" to the array vv:

»vv=load('myfile1.txt');
```

To import data from an ascii, tab delimited file that is *sans* extension use:

```
% save the data from the file "myfileXYZ" to the array vXYZ:

»vXYZ=load('myfileXYZ');
```

Direct importation of ***Excel*** *'.xls'* or *'.wbk'* files will NOT work by the methods noted above. Use the direct *cut & paste* approach between selected ***Excel*** workbook or worksheet files and the ***MATLAB*** array editor.

System Polynomials

MATLAB polynomial manipulations of $G(s) = \dfrac{3s^2 + 5s + 1}{s^2 + 1s + 1} = \dfrac{n(s)}{d(s)}$ are done as follows:

<u>To find polynomial factors:</u>

```
>> n=[3 5 1];
>> roots(n)
ans =
   -1.4343
   -0.2324

>> d=[1 1 1];
>> roots(d)
ans =
  -0.5000 + 0.8660i
  -0.5000 - 0.8660i
```

<u>To multiply two polynomials *n(s)*d(s)* use the "convolution" command:</u>

```
>> conv(n,d)

ans =
     3     8     9     6     1
```

which means $n(s) * d(s) = 3s^4 + 8s^3 + 9s^2 + 6s + 1$

<u>MATLAB command-line help yields the following info:</u>

```
>> help conv
```

CONV Convolution and polynomial multiplication.
C = CONV(A, B) convolves vectors A and B. The resulting
vector is length LENGTH(A)+LENGTH(B)-1.
If A and B are vectors of polynomial coefficients convolving
them is equivalent to multiplying the two polynomials.

See also DECONV, CONV2, CONVN, FILTER and, in the Signal
Processing Toolbox, XCORR, CONVMTX.

<u>Note the "See also" is particularly helpful since it reveals the following:</u>

```
>> help deconv
```

DECONV Deconvolution and polynomial division.
[Q,R] = DECONV(B,A) deconvolves vector A out of vector B. The result

is returned in vector Q and the remainder in vector R such that
B = conv(A,Q) + R.

If A and B are vectors of polynomial coefficients, deconvolution is equivalent to polynomial division. The result of dividing B by A is quotient Q and remainder R.

See also CONV, RESIDUE.

<u>Here we can easily perform "long division" of *n(s)/d(s)* as follows:</u>

```
>> deconv(n,d)

ans = 3
```

Where this means the division yields the "quotient" factor "3" plus an unspecified "remainder".

<u>To get explicit quotient *Q* and remainder *R* use the explicit output form:</u>

```
>> [Q,R]=deconv(n,d)

Q = 3
R = 0      2     -2
```

which means $G(s) = \dfrac{3s^2 + 5s + 1}{s^2 + 1s + 1} = \dfrac{n(s)}{d(s)} = Q + \dfrac{R(s)}{d(s)} = 3 + \dfrac{2s - 2}{s^2 + 1s + 1}$

<u>To perform Partial Fraction Expansion (PFE) use the explicit output form:</u>

```
>> [r,p,k]=residue(n,d)

r =  1.0000 + 1.7321i
     1.0000 - 1.7321i

p =  -0.5000 + 0.8660i
     -0.5000 - 0.8660i

k =  3
```

which means

$$G(s) = \frac{r(1)}{s - p(1)} + \frac{r(2)}{s - p(2)} + k(s) = \frac{1 + 1.7321j}{s - (-0.5 + 0.866j)} + \frac{1 - 1.7321j}{s - (-0.5 - 0.866j)} + 3s^0$$

See textbook: Palm, "System Dynamics", McGraw-Hill, 2005, Section 3.8 for more details.